EXAMINER'S AMENDMENT

1. An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to the applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

2. An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

3. Authorization for this examiner's amendment was given in a telephone interview with Mr. Kenneth Eiferman on 2/5/2009.

4. Please replace all claims with the following:

1. (Currently Amended) A system for dynamically detecting potential race conditions in a program having at least one thread and one or more shared memory locations, the system comprising:

a processor, wherein the processor is adapted to:

initialize a virtual clock (C_t) respectively for each of the at least one thread;

initialize a set of candidate locks for each of at least one shared memory location (S_x);

initialize a set of locks (S_t) held for each of at least one thread;

initialize a set of concurrent thread segments (T_x) accessing each of the at least one shared memory location;

initialize an ordered set of thread segments (B_t) below a current thread, wherein the ordering is achieved using the virtual clock associated with each thread;

upon detection of a fork of a new concurrent thread (t1) by a prior thread (t):

increment the virtual clock associated with the prior thread (t);

initialize the virtual clock associated with the new concurrent thread (t1);

update the ordered set of thread segments (B_t) ~~below~~ before the prior thread (t) by forming a union of a current ordered set for the prior thread (t) and a singleton set having a thread segment <t, C_t> comprising a tuple of a thread identifier and a virtual clock time;

upon detection of a join call by the prior thread (t):

update the ordered set of thread segments (B_t) by forming a union of the ordered set of thread segments (B_t), a second ordered set of thread segments (B_t1) ~~below~~ before thread (t1) that do not belong to the prior thread (t), and a singleton set containing a current thread segment associated with the current thread (t1); and

upon detection of a read or write to a memory location (x), determining whether a potential race condition exists by updating a set of concurrent thread segments concurrently accessing the memory location (T_x) after the read or write using the updated ordered set of thread segments (B_t) from the detection of the fork or join call.

2. (Cancelled)

3. (Currently Amended) The system of claim 1 wherein the processor is further adapted to ~~upon detection of a read or write of a memory location (x) by a thread (t)~~:

~~update a set of concurrent thread segments concurrently accessing the memory location (T_x) after the read or write;~~

if a cardinality of the set of concurrent thread segments (T_x) is less than or greater than 1, set a current value of a lockset (S_t) to a current value of the set of candidate locks for the memory location (S_x);

if the cardinality of the set of concurrent thread segments is greater than 1, set a new value of the set of candidate locks for the memory location (S_x) to an intersection of a prior value of the set of candidate locks for the memory location (S_x) and the lockset (S_t);

if the set of candidate locks (S_x) is empty and a new value of cardinality of the set of concurrent thread segments (T_x) is greater than 1, reporting a potential race condition.

4-22. (Cancelled)

23. (New) A method for dynamically detecting potential race conditions in a program having at least one thread and one or more shared memory locations, the method comprising:

initialize a virtual clock (C_t) respectively for each of the at least one thread;

initialize a set of candidate locks for each of at least one shared memory location (S_x);

initialize a set of locks (S_t) held for each of at least one thread;

initialize a set of concurrent thread segments (T_x) accessing each of the at least one shared memory location;

initialize an ordered set of thread segments (B_t) below a current thread, wherein the ordering is achieved using the virtual clock associated with each thread;

upon detection of a fork of a new concurrent thread (t1) by a prior thread (t):

increment the virtual clock associated with the prior thread (t);

initialize the virtual clock associated with the new concurrent thread (t1);

update the ordered set of thread segments (B_t) before the prior thread (t) by forming a union of a current ordered set for the prior thread (t) and a singleton set having a thread segment <t, C_t> comprising a tuple of a thread identifier and a virtual clock time;

upon detection of a join call by the prior thread (t):

update the ordered set of thread segments (B_t) by forming a union of the ordered set of thread segments (B_t), a second ordered set of thread segments (B_t1) before thread (t1) that do not belong to the prior thread (t), and a singleton set containing a current thread segment associated with the current thread (t1);

upon detection of a read or write to a memory location (x), determining whether a potential race condition exists by updating a set of concurrent thread segments concurrently accessing the memory location (T_x) after the read or write using the updated ordered set of thread segments (B_t) from the detection of the fork or join call; and

outputting a result of race condition determination to the user.

24. (New) The method of claim 23 further comprising:

if a cardinality of the set of concurrent thread segments (T_x) is less than or greater than 1, set a current value of a lockset (S_t) to a current value of the set of candidate locks for the memory location (S_x);

if the cardinality of the set of concurrent thread segments is greater than 1, set a new value of the set of candidate locks for the memory location (S_x) to an intersection of a prior value of the set of candidate locks for the memory location (S_x) and the lockset (S_t);

if the set of candidate locks (S_x) is empty and a new value of cardinality of the set of concurrent thread segments (T_x) is greater than 1, reporting a potential race condition.

25. (New) A computer readable medium having stored thereon computer executable instructions for dynamically detecting potential race conditions in a program

having at least one thread and one or more shared memory locations, the instructions comprising:

initialize a virtual clock (C_t) respectively for each of the at least one thread;

initialize a set of candidate locks for each of at least one shared memory location (S_x);

initialize a set of locks (S_t) held for each of at least one thread;

initialize a set of concurrent thread segments (T_x) accessing each of the at least one shared memory location;

initialize an ordered set of thread segments (B_t) below a current thread, wherein the ordering is achieved using the virtual clock associated with each thread;

upon detection of a fork of a new concurrent thread (t1) by a prior thread (t):

increment the virtual clock associated with the prior thread (t);

initialize the virtual clock associated with the new concurrent thread (t1);

update the ordered set of thread segments (B_t) before the prior thread (t) by forming a union of a current ordered set for the prior thread (t) and a singleton set having a thread segment <t, C_t> comprising a tuple of a thread identifier and a virtual clock time;

upon detection of a join call by the prior thread (t):

update the ordered set of thread segments (B_t) by forming a union of the ordered set of thread segments (B_t), a second ordered set of thread segments (B_t1) before thread (t1) that do not belong to the prior thread (t), and a singleton set containing a current thread segment associated with the current thread (t1);

upon detection of a read or write to a memory location (x), determining whether a potential race condition exists by updating a set of concurrent thread segments concurrently accessing the memory location (T_x) after the read or write using the updated ordered set of thread segments (B_t) from the detection of the fork or join call; and

outputting a result of race condition determination to the user.

26. (New) The method of claim 25 wherein the instructions further comprise:

if a cardinality of the set of concurrent thread segments (T_x) is less than or greater than 1, set a current value of a lockset (S_t) to a current value of the set of candidate locks for the memory location (S_x);

if the cardinality of the set of concurrent thread segments is greater than 1, set a new value of the set of candidate locks for the memory location (S_x) to an intersection of a prior value of the set of candidate locks for the memory location (S_x) and the lockset (S_t);

if the set of candidate locks (S_x) is empty and a new value of cardinality of the set of concurrent thread segments (T_x) is greater than 1, reporting a potential race condition.

5. Any inquiry concerning this communication or earlier communications from the examiner should be directed to MengYao Zhe whose telephone number is 571-272-6946. The examiner can normally be reached on Monday Through Friday, 10:00 - 8:00 EST. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Meng-Ai An can be reached at 571-272-3756. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

/Meng-Ai An/

Supervisory Patent Examiner, Art Unit 2195